# Application Note - AN011

# Driving CLKREQ, WAKE, PERST and similar on PCIe Modules

# October, 2016

Contents

[Driving CLKREQ, WAKE, PERST and similar on PCIe Modules 1](#_Toc464042088)

[October, 2016 1](#_Toc464042089)

[Introduction 3](#_Toc464042090)

[Requirements 4](#_Toc464042091)

[Set-up 4](#_Toc464042092)

[Control Options 4](#_Toc464042093)

[Signal Control 5](#_Toc464042094)

[The Basics 5](#_Toc464042095)

[Driving Options – QTL1630/QTL1688 6](#_Toc464042096)

[Events that trigger driving 7](#_Toc464042097)

[Driving Examples 8](#_Toc464042098)

[Assert a signal for a set time 8](#_Toc464042099)

[Drive a signal into a set state 9](#_Toc464042100)

# 

# Introduction

The latest revisions of Quarch PCIe modules added the ability to drive certain sideband signals high and low, in order to force the device under test into specific states.

The signals that can be driven are different on each module, and dependant on the interface specification.

Some signals can be driven both high and low, while some signals are limited. Again this depends on the hardware specification of the interface. For full details on the driving options for each signal, please see the relevant technical manual for the module.

Signal driving is an advanced feature, and is recommended to be used by engineers who understand the electrical characteristics of the interface.

## Requirements

1. Quarch PCIe Module which supports driving:

* QTL1630 (-04 and higher)
* QTL1688 (-03 and higher)
* QTL1743 (-02 and higher)
* QTL2034 (All versions)

1. Host PC (with drivers installed, if using USB control)
2. An appropriate script or terminal able to connect and issue commands

## Set-up

Host PC

Quarch Controller + PCIe Module

USB/Serial/LAN

## 

### Control Options

* TorridonTerminal can send commands to a module  
  <http://quarch.com/file/torridon-terminal>
* TestMonkey provides GUI control over a module  
  <http://quarch.com/file/testmonkey>
* Python can be used to automate control of Quarch products  
  <http://quarch.com/file/an-006-python-control-quarch-modules>

## Signal Control

### The Basics

Each ‘Signal’ (pin on the connector that can be controlled) can be in one of two states

* ‘Closed’ (Connected): The two sides are connected as if the Quarch module was not present
* ‘Open’ (Disconnected): The two sides are entirely disconnected (open circuit)

Each signal that can be enabled for driving has additional options:

* ‘Drive Open’: The driving action to be performed when the signal is in the ‘Open’ mode
* ‘Drive Closed’: The driving action to be performed when the signal is in the ‘Closed’ mode

If a Drive Open or Drive Closed option is set, then instead of just opening or closing the switch, we will instead actively drive the signal

Each of the drive (Open and Closed) settings can be set to one of three options

* HIGH Drive the signal high
* LOW Drive the signal low
* NONE Disable driving

### Driving Options – QTL1630/QTL1688

Every module has a different set of signals that can be driven. Each signal has some differences on what the driving actually does, depending on the electrical specification of the interface.

Below is an example of the PCIe x16 slot modules. Note that some options are marked as ‘Not Driven’. This means the signal is allowed to float, but will float to its own level.

These tables can be found in the Technical Manual for each device.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signal Name** | **Signal Type** | **Host Side Behaviour** | | **Device Side Behaviour** | |
| **High** | **Low** | **High** | **Low** |
| PERST# | Push/Pull output from Host | Not Driven | Not Driven | Driven High | Driven Low |
| CLKREQ# | Bidirectional Open Drain | Not Driven | Driven Low | Not Driven | Driven Low |
| WAKE# | Open Drain output from device | Not Driven | Driven Low | Not Driven | Not Driven |

### Events that trigger driving

We need to consider that there are various actions that will change signals between the open/closed states.

1. Setting a signal to source 0 will always disconnect the signal. Setting it to source 8 will always connect the signal
2. Using a Plug/Pull command will run the hot-plug logic, changing the state of every signal assigned to sources 1-7
3. Pin-bounce (including custom user pin-bounce) can be assigned to a sources 1-7, and can be used to create complex connect/disconnect patterns during a hot-swap event
4. Glitch commands can be used to change the current state of the signal for a very precise time. Modules with triggering can sync the glitch with external events.

If any of these events occur on a signal that has driving options set, then the signal will be driven. The drive state will change every time the open/closed state changes.

Full details of these actions are in the Technical Manuals for the products,

## Driving Examples

### Assert a signal for a set time

Perhaps we want to assert the PERST signal while the drive is operating in a normal state. To do this we need to have the drive up and running. This will be the default state of the module, or if powered down, we issue a power up command:

***RUN:POWER UP***

Asserting PERST requires us to assert (drive low) the signal for at least 100uS.

To do this, we will enable Glitching on the PERST signal

***SIGnal:PERST:GLITch:ENAble ON***

We now set the glitch engine to produce a 100uS glitch (500nS x 200 will do for this)

***GLITch:SETup 500ns 200***

Next, we set the driving feature to drive the signal LOW when the switch would otherwise be open (which will happen during the 100uS that the glitch event runs for)

***SIGnal:WAKE:DRIve:OPEn LOW***

Finally we run the glitch, which will immediately assert PERST for the required time

***RUN:GLITch ONCE***

### Drive a signal into a set state

We may want to drive CLKREQ low, to simulate a short circuit failure. Assuming we are in a default state, with the drive powered up and running:

The switches are currently closed (as the drive is running). We need to set the ‘Drive Open’ state, so that when we change the switch state, the signal is driven low:

***SIGnal:CLKREQ:DRIve:OPEn LOW***

Now we drive the signal, by forcing it into the open state. This is done by assigning it to source 0 (The ‘always off’ source). When assigned here, the signal will immediately move to the Open state.

***SIGnal:CLKREQ:SOURce 0***

The CLKREQ signal will now be driven low on both the host and device sides. To revert it back to normal again, we just assign CLKREQ back to a connected source:

***SIGnal:CLKREQ:SOURce 1***